CLAIMS

We claimed:

1. A computer implemented method comprising:

beginning initialization a first thread from a second context, wherein the first thread is capable of being executed on a first context and the second context;

suspending the initialization of the first thread at a position within the second context;

creating a second thread based on the position in the second context; and

completing the initialization of the first thread continuing from the position in the second context.

2. The method of claim 1, wherein the beginning initialization of the first thread includes allocating per-thread context resources.

3. The method of claim 1, wherein the beginning initialization of the first thread is suspended in response to a detection of an operating system request to create the first thread.

4. The method of claim 1, wherein the second context is the platform-independent code to be executed on a host platform.

5. The method of claim 1, wherein the second context is a host platform that supports multiple instruction set architectures (ISA).

6. The method of claim 1, wherein completing the initialization of the first thread includes executing an application programming interface that makes an operating system request to create the first thread in the second context.

7. The method of claim 1, wherein the first context is an IA-32 multithreaded program and the second context is an IA-64 multithreaded program.

8. The method of claim 1, wherein the first context is an IA-32 platform and the second context is an IA-64 platform.

9. A computer implemented method comprising:

beginning initialization a foreign thread from a host platform, wherein the foreign thread is to be executed on a foreign platform;

suspending the initialization of the foreign thread at a position within the host platform;

recording the position of the suspension;

creating a host thread from a host platform based on the recorded position; and

completing the initialization of the foreign thread continuing from the recorded position in the host platform.

10. The method of claim 9, wherein the beginning initialization of the foreign thread includes allocating per-thread context resources.

11. The method of claim 9, wherein the beginning initialization of the foreign thread is suspended in response to a detection of an operating system request to create the foreign thread.

12. The method of claim 9, wherein the host platform supports platform-independent code.

13. The method of claim 9, wherein the host platform supports multiple instruction set architectures (ISA).

14. The method of claim 9, wherein the foreign platform is a IS-32 platform and the host platform is a IS-64 platform.

15. A computer system for managing thread resource comprising:

a first multithreaded programming environment;

a second multithreaded programming environment;

a multithreaded program including a first thread and a second thread;

a host platform; and

a dynamic binary translator to manage and support the first thread for the first multithreaded programming environment to be executed in the second multithreaded programming environment.

16. The system of claim 15 further comprises a first component to provide a communication interface between the dynamic binary translator and the multithread programming environment.

17. The system of claim 15 further comprises a first thread library and a second thread library.

18. The system of claim 15 further comprises a second component to intercept service requests from the multithreaded program.

19. A computer system for managing thread resource comprising:

a random accessed memory;

a first processor capable of executing multithreaded programs stored in the random accessed memory;

a multithreaded program to be executed on a second processor; and

a program to transparently initialize and create a thread included in the multithreaded program in an environment supported by the first processor, to be executed on the second processor.

20. The system of claim 19, wherein the program further allocates per-thread context resources.

21. The system of claim 20, wherein the program initializes and creates the thread transparently by associating the allocated per-thread context resource between the environment supported by the first processor.

22. A machine-accessible medium that provides instructions that, when executed by a processor, causes the processor to:

beginning initialization a first thread from a second context, wherein the first thread is capable of being executed on a first context and the second context;

suspending the initialization of the first thread at a position within the second context;

creating a second thread based on the position in the second context; and

completing the initialization of the first thread continuing from the position in the second context.

23. The machine-accessible medium of claim 22, wherein the beginning initialization of the first thread includes allocating per-thread context resources.

24. The machine-accessible medium of claim 22, wherein the beginning initialization of the foreign thread is suspended in response to a detection of an operating system request to create the foreign thread.

25. The machine-accessible medium of claim 22, wherein the first context is an IA-32 multithreaded program and the second context is an IA-64 multithreaded program